平成26年度　卒業論文

携帯端末の画像処理による微小オブジェクトの追跡

平成27年2月19日

111108121

山口 陽介

指導教員 木村　広 准教授

九州工業大学 工学部 電気電子工学科

目 次

第 1 章　序論 **4**

1.1　はじめに 4

第 2 章　関連研究と本研究の位置付け **6**

2.1　ドップラー方式のスピードガンを使った分析 6

2.2　ビデオカメラとコンピュータを使った分析 7

2.3　加速度センサを使った分析 7

2.4　レーザー装置を使った分析 7

2.5　ボールの到達時間からの分析 8

2.6　移動物体を指でなぞるアプリケーションでの分析 8

2.7　iPhoneSG の位置づけ 9

第 3 章　開発環境 **10**

3.1　開発プラットフォーム 10

3.1.1　OpenCV のインストール 11

3.1.2　CocoaPods のインストール 12

3.2　開発のベースプログラム 13

3.3　開発ターゲット 13

第 4 章　本システムのおこなう画像処理 **14**

4.1　フレームレート 120fps,240fps での撮影 14

4.1.1　精度の向上 15

4.1.2　ノイズの低減 16

4.1.3　画質の問題 16

4.2　ボールの探索・検出 16

4.2.1 動画から静止画像の切り出し 17
4.2.2 背景除去 17
4.2.3 平滑化 18
4.2.4 輪郭抽出 20
4.3 ボールの位置予測 20
4.4 球速の計算 22

第 5 章 結果と考察 **24**
5.1 アプリケーションの使用方法 24
5.2 実験方法 25
5.3 測定の条件 25
5.4 測定結果 26

第 6 章 まとめ **28**

付 録 A Appendix **31**
A.1 AppDelegate 31
A.2 ViewController 33
A.3 AVCaptureManager 47

第1章　序論

1.1　はじめに

本研究では、携帯端末 iPhone6 上で、コンピュータビジョンの技術を駆使し、野球でもっとも注目されるピッチャーの投球速度を計測するアプリケーションを開発する。150km/h にもなる速度を持つボールという小物体を追跡し、その速度を計測するスピードガンアプリケーションを、CPU パワーが十分とは言えない、いわゆるスマートフォン上で実現することに挑戦し、かつ、従来型のスピードガンとは違った利用方法、利用形態を利用者に提供することで、野球技術の向上とともに、野球観戦の楽しみを広げることを目指した。

プロ野球の観戦においては、球場に設置されたスピードガンによる球速の表示は今では当たり前となっている。しかし、高校野球の試合や練習中の球速の測定はまだまだ一般的ではない。球場設置のスピードガンによる測定は、福岡県では県大会ベスト８以上の試合でしかサービスされず、市販のスピードガンを手に持って測定するには、測定する位置がピッチャーとキャッチャーの延長線上のバックネット裏に制限される。だれもが知りたいときに知れるものではない。ビデオカメラで撮影した動画を、自宅あるいは研究室に持ち帰り、解析することも可能だが、「その場」で球速を知りたいという戦術戦略上の理由、野球観戦上の楽しみという観点からは十分なものではない。

一方、近年、スマートフォンは携帯電話としてだけでなく、小型のコンピュータとして利用できるスペックを持つようになっている。それとともにスマーフォンに実装されているカメラの機能も進化してきている。とくに、iPhone シリーズの最新機種である iPhone6 はフレームレート最大 240fps での動画撮影が可能となっている[1]。この iPhone 6 には実用上、十分な精度のスピードガンを搭載できる可能性がある。

一般のいわゆるスピードガンは、電磁波や超音波が対象物にぶつかり、反射する際の

[1]iPhone は日本で最も多くの人に利用されている。2014 年 12 月の株式会社ウェブレッジによる調査では iPhone のシェア率 53.52 % という結果がある。2 位は Xperia A の 2.89 % である

ドップラー効果を利用して速度を計測するものがほとんどである。それに対し、iPhone6に搭載のカメラ、CPU による画像処理でそれと同等、あるいは、別角度からの新しい「スピードガン」の開発を本研究では目指した。

本研究では開発の目標を次のように設定した。

- 撮影から画像処理、出力までを iPhone のみで完結させる。
- iPhone を手に持って測定できる。
- 12 秒以内に処理できる。[2]
- 120km/h～150km/h の測定をプラスマイナス 5km/h 以内の精度でおこなうことができる。

以上の目標が達成出来れば、従来のスピードガンと同等以上の機能をもつアプリケーションがスマートフォン上のアプリケーションとして実現できたと判断することができる。

[2]公認野球規則 [12] によれば、ピッチャーはキャッチャーからボールを受けて、12 秒以内に次の投球を開始しなくてはならないというルールがある。違反した場合、試合を長引かせたとし、審判はボールを宣告する

第2章　関連研究と本研究の位置付け

野球の投球を分析する研究はこれまでにも様々な方法でおこなわれてきている。

2.1　ドップラー方式のスピードガンを使った分析

球速を測定する代表的な機材として、電磁波や超音波のドップラー効果を利用したスピードガンがある。蔭山らの研究 [16] や勝亦らの研究 [20] のように、投球を分析する研究でもよく用いられている。

ドップラー方式のスピードガンの場合、まずはコストの問題がある。60m の距離まで測定可能で観客席からも測定が可能なハイグレードタイプ[1]のものや測定距離 30m までのもの[2]、近距離でのみ使えるもの[3]等、様々な種類のスピードガンがあるが、高価であり、ファンや選手が気軽に利用できるものではない。これがスピードガンを利用する人が少ない原因の一つだと考えられる。

また、測定精度の問題もある。ドップラー方式のスピードガンはボールに向けて電磁波を照射し、ボールからの反射波を測定する。ドップラー効果により照射波と反射波では周波数に変化がおきているため、これらの波の周波数を比較することにより球速を計算している。正しい測定結果を得るためにはボールの軌道上にスピードガンを固定しておく必要があり、ボールの軌道に対して角度があるとその分だけ誤差が生じる。ハイグレードタイプのスピードガン HP-2[13] の場合、球速 100km/h のボールを測定したとき、ボールの投球軌道からの計測角度が 10 度で 98.48km/h、20 度で 93.69km/h、40 度で 76.60km/h と表示される。測定はキャッチャーの真後ろからおこなう必要があるため、測定をおこなえるのはバックネット裏の一部の人のみに限られてしまう。バックネット裏

[1] ミズノ スピードガン HP-2 2ZM1050：520,000 円
[2] ミズノ スピードガン YUPITERU：102,600 円
[3] BUSHNELL デジタルスピードガン スピードスター V：20,475 円

で測定したとしても上方向に角度ができることを考慮する必要があり、正確な測定は難しい。

2.2 ビデオカメラとコンピュータを使った分析

ビデオカメラを投球の分析に利用した研究の例としては、二台の高速度ビデオカメラを使って撮影をおこない、投球を分析した奥村 [15] や島田ら [21] の研究や、周りを暗くした地下室においてストロボ撮影によって投球を分析した Eloy Durn ら [10] の研究、光学式モーションキャプチャシステム MAC3D を用いて投球を分析した蔭山ら [14] の研究など、様々な種類、個数のビデオカメラでの研究がおこなわれている。

高速ビデオカメラは高価で、分析のための専用の空間を用意するのも難しい。研究でおこなわれているような分析は手軽とは言えない。また、いずれの研究もビデオカメラで撮った動画をコンピュータに渡して投球の分析をおこなっている。撮影した動画を研究室に持ち帰って分析するため、その場で球速を知ることはできない。

2.3 加速度センサを使った分析

加速度センサを使った分析の例として、手首に加速度センサを取り付け、その波形からボールリリース前後の投球パターンの分類をおこなった斎藤健治らの研究 [19] がある。また、Apple Store では、「びゅん」[6] のように、携帯端末をボールを投げるように振り、端末内の加速度センサによって球速を推定するアプリケーションが配信されている。

このような測定方法の場合、取り付ける装置が高価であったり、投球の邪魔になってしまうことが問題となる。携帯端末を振る場合、端末を投げてしまう危険がある上、実際にボールを投げて計測することはできない。

2.4 レーザー装置を使った分析

斎藤健治らの研究 [19] では、レーザー装置を使った球速の測定もおこなっている。レーザー装置は 1.0m×1.44m のアルミ製の枠の下段に 16 個の発光部、上段に 16 個の受光部をそれぞれ 5cm 間隔で並べ、さらに投球方向に 0.45m 離れた位置に同じ設定でレー

ザーを並べた二重カーテン状の装置である。この二重のレーザーカーテンを通過する時間を計測し、球速を算出している。

この方法を利用する場合、装置の用意が難しい。投球の経路上に装置を設置するため、試合中の測定は不可能で、測定用に専用の空間を用意する必要がある。また、測定が可能なのはこの装置の内側を通ったボールに限られる。

2.5 ボールの到達時間からの分析

Apple Store では、「Easy SpeedGun」[11] のように、ピッチャーがボールが投げたらスマートフォンの画面をタッチ、キャッチャーがボールを捕球したらもう一度タッチ、または投げたらタッチしてボールが捕球されたら指を放す、という操作でボールの到達時間を測り、球速を算出する、という手法のアプリケーションが配信されている。

スマートフォンを使って測定ができ、非常に手軽ではあるが、精度は期待できない。ピッチャーからボールが離れてキャッチャーまで到達するのにかかる時間は、球速 100km/h で約 0.6 秒、120km/h で約 0.5 秒、150km/h で約 0.4 秒である。この測定方法ではタッチのタイミングが 0.1 秒ずれると測定結果が大きく変わってしまう。人によってタッチのタイミングも違い、正確な測定をおこなうことは難しい。

2.6 移動物体を指でなぞるアプリケーションでの分析

野球用ではないが、Android アプリケーション「スピードガン - Speed Gun」[2] では、動いている物体をタッチし、その動きに合わせて指で物体をなぞり、その速度を計算する。

車のように大きさの大きい物体であれば、指でなぞりやすく、正確になぞることができれば高い精度の結果が得られるかもしれないが、野球の投球に応用する場合、ボールの動きが速く大きさも小さいため、指で追うのは難しいだろう。

2.7　iPhoneSG の位置づけ

本章で上げた関連研究では、特殊な機材を用いたり、分析用に空間を用意する必要があった。スマートフォンアプリケーションでおこなう場合、精度が人間任せになってしまうものが多い。

iPhoneSG は野球の試合や練習中のボールの動きをグラウンドや観客席から撮影し、画像処理することにより、ボールを自動で検出し、球速の推定をおこなう。測定に特殊な準備をする必要はなく、データを研究室に持ち込み、専用の画像処理を施す必要もない。ユーザがおこなうのは、測定できる位置で測定開始のボタンを押すのみである。

第3章　開発環境

3.1　開発プラットフォーム

iPhoneSG の開発にプラットフォームは Apple 社製 iMac(3.5GHz, RAM 12GB) を用いた。

利用したソフトウェアは表 3.1 のとおりである。

表 3.1: 開発に使用したソフトウェア

ソフトウェア	バージョン	配布元
Xcode	6.0.1	https://developer.apple.com/jp/xcode/downloads/
OpenCV	2.4.9	http://opencv.org
CocoaPods		http://cocoapods.org

Xcode[1] は Apple 社が提供するエディタ、ライブラリ、コンパイラ、デバッガ、シミュレータが統合された開発環境である。MacOSX、iPhone、iPad のアプリケーションを開発できる。

Xcode にも Apple 独自の画像処理ライブラリが付属する。しかし、本研究では Apple ライブラリの利用を内蔵カメラからの画像入力、処理中・処理後のディスプレイへの画像出力のみとし、コアとなる画像解析は OpenCV[5] のフレームワークを利用した (図 3.1)。OpenCV は MacOSX 以外、Windows や Linux の上でも広く使われ、実績もあり、資料も豊富である [18], [7]。

Xcode の開発言語は Objective-C、OpenCV の開発言語は C++ と異なる。このままでは iPhone のカメラで撮影した映像を OpenCV で解析できない。しかし、CocoaPods[3] を利用することで、Objective-C の文脈から OpenCV の C++ ライブラリを自由に呼び出せるようになる。本研究でも CocoaPods は必須のツールである。

Xcode、OpenCV、CocoaPods はいずれもフリーなソフトウェアであり、インターネットからダウンロードし、インストール、実行することができる。

一般に、iPhone アプリケーションの開発は、

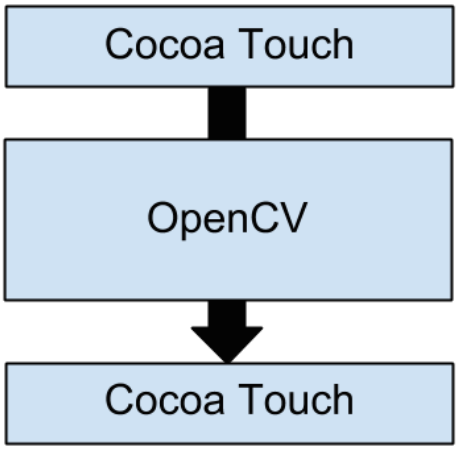

図 3.1: iPhoneSG による画像処理の流れ。Cocoa-Touch から OpenCV にデータを渡す。

1. MacOSX 上の Xcode でプログラム開発をおこない、
2. その動作を Xcode のデバッガ、シミュレータで確認後、
3. 開発したプログラムを iPhone へ転送する

という手順になる。本研究もこれにしたがう。

3.1.1 OpenCV のインストール

OpenCV のインストールは、OS X 用のパッケージ管理システム MacPorts[8] を利用した。

ターミナルを立ち上げ、MacPorts のコマンドを入力してインストールする。OpenCV の情報を確認するためには、ターミナルを開き、次のコマンドを実行する。

```
$ port info opencv
```

これで最新のバージョンやライセンスなど、OpenCV に関する情報を見ることができる。OpenCV をインストールする前に、OpenCV が必要とするライブラリをインストールする必要がある。次のコマンドを実行する。

```
$ sudo port install plgconfig zlib
```

このコマンドが完了したら、OpenCV のインストールをおこなう。OpenCV のインストールには、次のコマンドを実行する。

```
$ sudo port install opencv
```

これで OpenCV を利用する準備ができた。

3.1.2 CocoaPods のインストール

iPhone のアプリケーション開発に OpenCV を導入するために Objective-C のライブラリ管理ツール CocoaPods を利用した。CocoaPods のインストールには、Ruby 言語用のパッケージ管理システム RubyGems を利用する。ターミナルを開き、次のコマンドを実行する。

```
$ sudo gem install cocoapods
$ pod setup
```

これで、CocoaPods を利用する準備ができた。

CocoaPods を使い、iPhone のアプリケーション開発に OpenCV を導入するには次のような手順でおこなう。

1. Xcode で iOS プロジェクトを作成
2. Xcode を一旦終了
3. プロジェクトのルートディレクトリに Podfile という名前で空のファイルを作成
4. Podfile に pod 'OpenCV' を記述
5. 同ディレクトリで $pod install を実行
6. Xcode で .xcworkspace のファイルを開く
7. OpenCV を利用したいファイルの拡張子を .m から .mm に変更

これで、OpenCV のライブラリを iOS から利用することができる。

3.2 開発のベースプログラム

iPhoneSG のプログラムは Shuichi Tsutsumi の開発した SlowMotionVideoRecorder[9] をベースとした。SlowMotionVideoRecorder には以下の特徴がある。

- 120fps での高速撮影を実装
- カメラ機能は AVFoundation フレームワークを利用
- フレームレート 30fps /60fps /120fps の切り替えが可能

3.3 開発ターゲット

iPhoneSG の開発ターゲットは iPhone6 である。

2015 年 1 月の株式会社ウェブレッジによる調査 [17] ではスマートフォン市場における iPhone のシェア率が 50% を超えているという結果がある。iPhone6 は 2014 年 10 月の開発開始時点で iPhone シリーズの最新機種のスマートフォンであり、人気も高い。iPhone6 のハードウェアスペックを表 3.2 に示す。CPU クロック等、一部のデータは未公表であり、信頼のおける専門誌 [4] による推定値（*で表示）を引用している。

比較のため、iPhone5s、Android 端末 Xperia Z3 のハードウェアスペックも示している。

表 3.2: iPhone6, iPhone5S, Xperia Z3 スペック比較（*は推定値）

端末名	iPhone6	iPhone5s	Xperia Z3
CPU クロック	1.4GHz*	1.3GHz*	2.5GHz
メモリ	1GB*	1GB*	3GB
動画撮影	1080p/60fps, 720p/240fps	1080p/30fps, 720p/120fps	1080p/30fps, 720p/120fps

動画に映る小さな白球のイメージを検索し、精度高くその移動速度を得るためには、高いフレームレート、十分な CPU パワーが必須である。iPhone6 は 240fps での動画撮影が可能となっており、CPU の処理能力も旧機種の iPhone5S に比較し 2 倍（推定値）とされる。

第4章　本システムのおこなう画像処理

本研究の大きな特徴は iPhone のみで画像処理をおこなう事である。PC に比べ、処理能力の劣るこのデバイスで開発するにあたって、処理時間は大きな問題となる。野球にはランナーがいないとき、ピッチャーは 12 秒以内に次の投球を開始しなくてはならないというルールがあり、処理時間には制限がある。本章では、iPhoneSG がおこなう各処理に加え、高速化・手ぶれに対するアプローチをこの章で説明していく。

iPhoneSG のおこなう大まかな処理の流れを図 4.1 に示す。

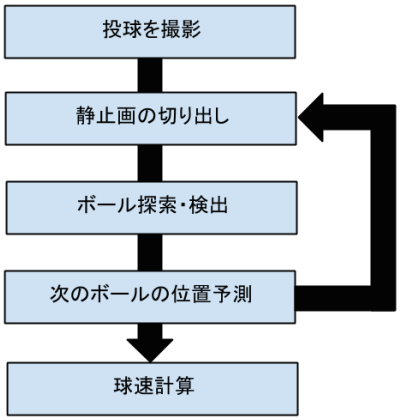

図 4.1: iPhoneSG がおこなう処理の流れ

4.1　フレームレート 120fps,240fps での撮影

今回のアプリケーションでは、iPhone6 の標準カメラアプリのスローモーション撮影でも採用されている 120fps、240fps の 2 種類のフレームレートの動画を扱うことにした。この 2 種類のフレームレートを選ぶことにより、各フレームレートでの撮影部分を開発する前の段階でも、標準カメラアプリで動画の撮影をおこなうことで、解析の対象と

なる 120fps、240fps の動画が用意できる。これにより、iPhone6 に比べてハイパワーな iMac を使って動画解析部分の開発を並行しておこなうことができる。

iPhone6 の標準カメラアプリではフレームレート 30fps、60fps での撮影も可能だが、これらのフレームレートでは球速の測定精度が目的の達成には不十分と判断し、120fps、240fps での撮影を利用することにした。

iPhone6 でフレームレート 120fps、240fps の動画を利用する際に、次のような利点、問題が出てくる。

- 精度の向上
- ノイズの低減
- 画質の問題

4.1.1 精度の向上

高いフレームレートで撮影する大きな利点はボールの動きをより細かく撮れることである。60fps と比較したとき、120fps なら 2 倍、240fps なら 4 倍のフレーム数で撮影できるので、その分だけボールの動きは細かくなる。

球速を計算するために iPhoneSG のプログラムでは、探索範囲内でボールが検出されたフレームをカウントしていくのだが、60fps、120fps、240fps、それぞれのフレームレートで同じ速度のボールを撮影した時でも結果は違ったものになってくる。各フレームレートでカウントされたフレームの数がひとつ違う時の球速の違いを表 4.1 に示す。

表 4.1: 各フレームレートの精度

フレームレート	60fps			120fps			240fps		
ボールが検出されたフレーム数	29	30	31	59	60	61	119	120	121
球速 [km/h]	137.3	132.8	128.5	135.0	132.8	130.6	133.9	132.8	131.7

表 4.1 に示したとおり、60fps の時はカウントしたフレーム数が 1 つ多いと球速が 5km/h 程度遅くなってしまうが、120fps ではフレーム数 1 つの差で球速が 2〜3km/h、240fps では 1km/h 程度の変化になる。つまり、60fps での測定では 5km/h よりも細かな測定はできないが、240fps であれば実際の球速に近い値を得ることができる。

4.1.2　ノイズの低減

ノイズの発生原因は、カメラを手で持って撮影する際の手ぶれである。撮影中に手ぶれが起こると画面全体にずれが生じる。このずれがフレーム間で差分をとったときにノイズとなる。ノイズの発生は手に持って撮影する以上、仕方のないものであるが、60fps の時の差分画像と高フレームレートの時の差分画像では、高フレームレートの時のほうが確実にノイズを小さくできる。本アプリケーションで用いたフレームレート 120fps、240fps の場合を考えていくと、60fps の時に生じるずれを、2 倍、4 倍のフレーム数で表現することになる。仮に 60fps の時のフレーム間で 4 ピクセルのずれが生じた場合、フレーム間でのずれは、120fps なら 2 ピクセル、240fps なら 1 ピクセルに小さくなるはずである。

4.1.3　画質の問題

iPhone6 のカメラにおいて、フレームレート 60fps までの動画の撮影と 120fps、240fps の動画の撮影では解像度に違いがある。60fps の動画撮影では iPhone6 の画面全体を 1920×1080 ピクセルの画像で表現するのに対し、120fps、240fps では 1280×720 ピクセルで表現する。これは iPhone6 の処理能力において、撮影のフレームレートを上げるためには画質を落とす必要があるためである。

画像全体の画素数が少なくなると、ボールを探す速度は速くなるが、ボール自体の画素数も小さくなってしまう。iPhoneSG のアルゴリズムではノイズの除去のために平滑化をおこなっているが、60fps の時に比べ、ボールとノイズのサイズ差が小さくなってしまうため、ボールまで除去してしまわない配慮が必要である。

4.2　ボールの探索・検出

ここからは iPhoneSG でボールを探索・検出する際におこなう画像処理について説明していく。処理の対象は、iPhone 6 で撮影されたフレームレート 240fps、120fps それぞれ 2 秒間の動画とする。これから説明する各画像処理は OpenCV の関数を使っておこなっている。

4.2.1　動画から静止画像の切り出し

フレームレート 120fps、240fps での撮影中にオンメモリで画像処理をおこなうのに、iPhone6 ではスペック不足であった。そのため、投球の撮影と画像処理を同時並行ではおこなわず、iPhone のカメラで投球を撮影し、それを一旦 mp4 ファイルとして iPhone 本体に保存する処理をおこなっている。iPhone 内に保存された mp4 ファイルから毎フレームの静止画を取り出していき、それらを処理をしていく。

図 4.2: 静止画像の切り出し

4.2.2　背景除去

高速で移動している白いボールを取り出すために背景の除去をおこなう。ここでは、画面の中で動きのない部分、つまり、ボールを除く画面全体を背景と呼んでいる。

この背景を除去し、動いているボール部分のみを残すために、まず図 4.3、4.4 のような連続した 2 フレーム間での差分を取る。変化のない部分は画素値が同じなので差分をとると 0 [1]になる。iPhoneSG では、高速に動いているボールという小さな白い物体の検出をおこなうため、動いている部分のみが残るこの方法を採用した。

フレーム間差分後の処理画像に二値化をおこなう。

[1]OpenCV においてグレースケールでの 0 は明るさが 0 、つまり黒を表す

図 4.3: 引かれるフレーム

図 4.4: 引くフレーム

二値化は画像を白と黒の 2 色だけで表現した画像にする操作である。二値化の前段階として、グレースケール変換をおこなっている。グレースケール変換によって処理画像は黒（明るさ最小：0）から白（明るさ最大：255）までの明暗のみで表現される。二値化の方法としては、明るさの閾値を設定し、閾値以上の値は白（255）、それ以下の値はすべて黒（0）にする二値化を採用した。これにより、処理画像は図 4.5 のように白と黒の 2 色のみで表現される。

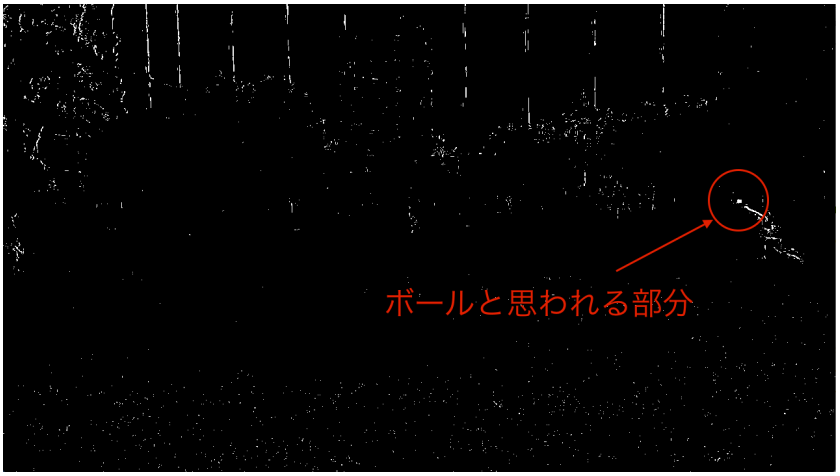

図 4.5: 差分後二値化処理画像

4.2.3 平滑化

平滑化は、画像をぼかしたり、画像中に発生したノイズを除去するときに用いられる手法である。

iPhoneSG は、手に持った状態での使用を想定しているため、撮影時に手ぶれが発生することは避けられない。手が動くときに背景も動き、フレーム間の差分を取ったときにノイズが発生する。高フレームレートの映像では、背景のズレの動きはボールの移動に比べて小さいため、ボールが消えない程度の平滑化をおこなうことでノイズを除去していく。平滑化には二値画像の収縮・膨張をおこなうことにする。収縮・膨張は白黒の二値画像の際に用いられる平滑化手法である。

収縮を 1 回おこなうと、周りに白の画素が無い注目画素は黒に置き換え、画像中の白の部分の面積は小さくなる。膨張はその逆で、1 回おこなうと、周りに一つでも白の画素があれば注目画素は白に置き換え、画像中の白の部分の面積は大きくなる。今回は収縮を 2 回おこなった後、膨張を 3 回おこない、ボール部分を強調した。図 4.5 に対してこの平滑化をおこなうと、図 4.6 のようにノイズが除去され、ボール部分が残る。収縮の回数を 2 回としたのは、探索対象である野球ボールの大きさが非常に小さいため、3 回以上収縮をおこなうとボールが完全に消えてしまい、1 回ではノイズが残ってしまうからである。ボールが消えないぎりぎりの大きさまで収縮をおこなっているため、収縮後のボール部分は白い小さな点に近い状態になる。これを膨張していくとボール部分は図 4.6 のように、矩形となって強調される。

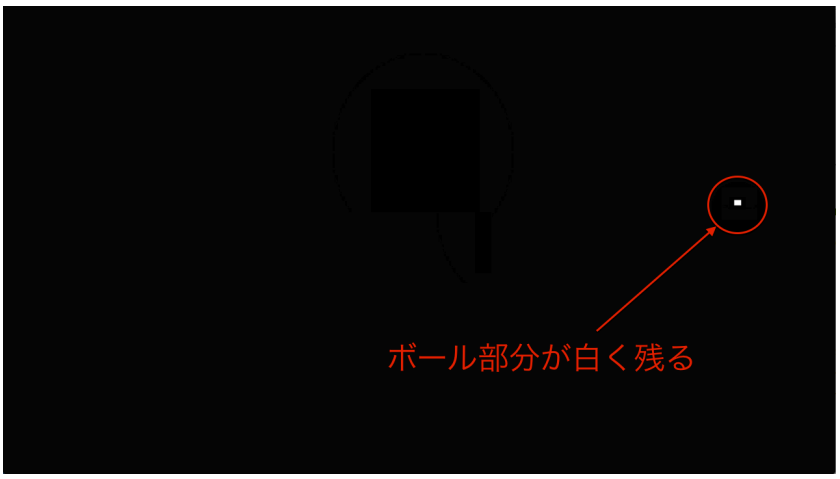

図 4.6: 収縮・膨張処理後の画像

4.2.4　輪郭抽出

これまでの処理によってボール部分を強調することはできたが、ボールの座標はまだ得られていない。輪郭抽出では白と黒の境目を輪郭と見て、白の面積がボールの大きさと同じくらいであればボールと判断する。面積が大きすぎる、または小さすぎる場合はボールだとは判断せず、そのフレームではボールは検出されない。

この時、輪郭の重心の座標からボールの位置座標を得ることができる。ここで得られた座標は次のボールの位置座標の予測に利用する。

4.3　ボールの位置予測

ハイパワーのコンピュータに比べ、メモリも CPU も限られるデバイスで目標である 12 秒以内の処理時間を実現するために、次のフレームでのボールの検出位置座標を予測し、探索範囲を限定することで、処理の高速化を図っている。探索範囲を限定するために探索対象の特徴を利用する。

探索対象である野球のボールには次のような特徴がある。

- 白くて丸い。
- 投球の画像中でボールは非常に小さい。
- 1 フレームに最大一つしか現れない。
- 高速に移動している。
- 等速直線運動をしている。
- マウンドからホームベースまでの距離は決まっている。[2]
- 撮影時間が短い。

探索範囲を限定していく流れを、一塁側からピッチャーの投球を撮影する、つまり画面の右から左へボールが移動している場合で説明していく。

[2]公認野球規則 [12] では、マウンドプレートからホームベースの先端までの距離は 18.44 m と定められている。

野球のルールからピッチャーのマウンドプレートからホームベースまでの距離は図 4.7 のように決まっている。iPhoneSG では、予め図 4.8 のようにマウンドプレートとバッターボックスの位置にマーカーを配置しており、そのマーカーにピッチャーとバッターが入るようにして撮影をおこなう。これにより、画面上での距離を決定することができる。

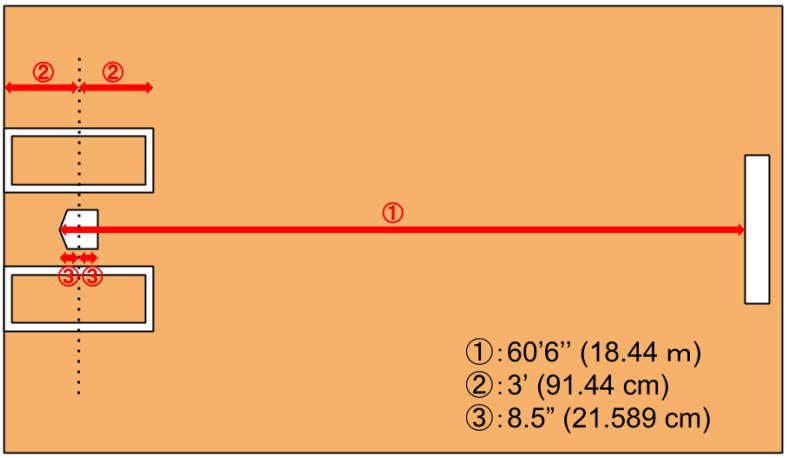

図 4.7: 野球の距離のルール

図 4.8: iPhoneSG の起動画面

ボールを探索する範囲全体は図 4.9 の ① のように設定している。バッターの立つ位置やピッチャーの踏み出し幅、リリースポイントは選手によって変わるため、ピッチャーとバッターは探索範囲から外している。これにより、ピッチャーやバッターによらない測定をおこなうことができる。

まずボールが最初に検出されるのは、画面の右端付近（図 4.9 の ②）だと考えられる。1 点目のボールが検出されるまでは右端の範囲のみを探索する。1 点目が検出された後はその座標から少し左にずれたところ（図 4.9 の ③）にボールが検出されるると予測できるので、探索範囲は 1 点目の座標の少し左になる。ボールが上方向や下方向に大きく移動することは考えにくいので、探索範囲の縦幅は 1 点目を探索するときよりも狭くしている。2 点目が検出されたら 3 点目以降は、1 点目と 2 点目の座標の差を取り、その差分だけ左にずれた範囲（図 4.9 の ④）を探索していけば良い。ボールの軌道が途中で大きく変化することはないので、この時の探索範囲は 1,2 点目よりも小さくできる。

次のボールの位置座標を予測することで、処理時間の短縮だけでなく、探索範囲以外のノイズを無視できるので、誤検出の防止にもつながる。

4.4 球速の計算

4.2、4.3 の処理を毎フレームおこなっていき、探索範囲内でボールが検出されたところでボールのカウントを開始、ボールが探索範囲の端に到達したところで探索、ボールのカウントを終了し、計算を開始する。球速は、

$$Verocity = \frac{Distance}{\frac{Count}{Framerate}} \times 3.6 \tag{4.1}$$

の式で計算している。（Velocity:速さ [km/h], Distance:投球距離, Framerate:フレームレート, Count:ボールが検出されたフレーム数）

図 4.9: 探索範囲の推移（①：探索範囲全体、②：1 点目の探索範囲、③：2 点目の探索範囲 ④：3 点目以降の探索範囲）

第5章　結果と考察

iPhone6 で撮影した動画から画像処理をおこなうアプリケーション iPhoneSG を使用して、球速の測定をおこなった。撮影や画像処理などのすべてを iPhone6 の端末上で完結する。

5.1　アプリケーションの使用方法

iPhoneSG では 120fps と 240fps、どちらかのフレームレートを選択することができる。

ピッチャー、バッターの位置にマーカーを配置しており、ユーザはこのマーカーの中にピッチャーとバッターが入るようにして撮影をおこなう。測定時の画面は図 5.1 のようになる。観客席やグラウンド上等、様々な場所で測定がおこなえるようにズーム機能も備えている。

図 5.1: 測定時の画面の様子

画面下方の赤い円が測定開始のボタンとなっており、ピッチャーがボールを放すときにこのボタンをタップし、測定を開始する。測定開始から 2 秒後に自動的に録画を終了し、ボールの検出処理をおこなう。ピッチャーがボールを投げてからキャッチャーが捕球するまでの到達時間は、時速 150 km で約 0.4 秒、時速 100 km で約 0.6 秒であり、ボールが移動している時間は非常に短い。そのため、1 秒程度の撮影時間でも計測は可能だが、ピッチャーのフォームや場面の違いによってボールを投げる瞬間にボタンを押せなくても測定がおこなえるように、余裕を持って 2 秒という撮影時間を設定した。これにより、ボールを離す瞬間ぴったりにボタンを押す、というユーザの負担は解消される。

5.2 実験方法

今回の測定実験は、フレームレート 120fps、240fps それぞれ 10 回ずつの計測を、九州工業大学運動場、野球場にておこなった。エラー等により、計測結果が出なかったものについては比較対象から外している。アプリケーションの測定結果との精度の比較として、測定時に撮影した動画を iPhone に保存しておき、研究室に戻って Mac 上でその動画の解析をおこなった。Mac 上では、撮影した動画を 1 フレームずつ見ていき、ボールが手を離れた瞬間からボールが捕球されるまでのフレーム数を人の目でカウントし、球速の計算をおこなった。

5.3 測定の条件

手ぶれによってボールが前のフレームよりも後方の位置に現れてしまうことがある。本アプリケーションでは、右から左に投げられるボールにおいて、ボールの位置座標が前のフレームよりも右にある場合、エラーを表示して処理を終了するようにしている。

また、天候によっても測定が成功しにくい場合があった。天気が晴れの場合、撮影者の真後ろに太陽がある順光のときが最も測定が成功しやすく、逆光や斜光のときのようにボールに影ができてしまう場合、ボールが検出しにくくなる。太陽が隠れている場合、ボールに影ができないため、測定は成功しやすいが、明るさが足りないと成功しにくくなる。検出できなかった場合は、測定結果は 0km/h となる。今回の測定は晴れた日に順光に近い方向でおこなっている。

5.4 測定結果

各フレームレートでおこなった測定の結果とその時の処理時間、Mac 上での速度の比較を表 5.1、5.2 に示す。

表 5.1: 測定結果（フレームレート 240fps）

iPhoneSG 測定結果 [km/h]	iPhoneSG 処理時間 [ms]	Mac 手作業 [km/h]
81	51489	81.98
71	36027	71.78
68	36037	68.06
109	9092	108.72
100	10054	98.55
91	35052	92.08
93	23429	92.08
71	8064	72.50
104	21102	102.73
112	17698	113.07

表 5.2: 測定結果（フレームレート 120fps）

iPhoneSG 測定結果 [km/h]	iPhoneSG 処理時間 [ms]	Mac 手作業 [km/h]
76	4924	74.88
59	4226	59.74
76	3079	76.93
78	4004	79.10
69	3723	72.94
82	3459	80.23
48	4003	46.41
79	2874	75.89
75	3491	75.89
47	4246	45.66

iPhoneSG と Mac 上での測定結果での平均誤差、最大誤差はそれぞれ表 5.3、5.4 のようになった。240fps では、平均誤差 0.94km/h、最大でも 1.5km/h と、精度の高い測定がおこなえたが、処理時間が 8~51 秒かかってしまい、目標であった 12 秒以内の処理は達成できなかった。120fps では、5 秒以内の処理時間を達成でき、精度も目標としていたプラスマイナス 5km/h 以内を達成できた。

120fps を利用する場合は野球の試合中でも十分に使える処理時間と精度を実現できた。240fps の場合は、投球間隔を気にしなくて良い練習中の投球など、利用シーンは限定されるが、より精度の高い測定をおこなうことができる。

iPhoneSG の開発開始が 10 月であり、プロ野球をはじめ、全ての野球のシーズンは

表 5.3: 測定誤差（フレームレート 240fps）

平均測定誤差 [km/h]	平均処理時間 [ms]	最大誤差 [km/h]	最長処理時間 [ms]
0.94	24804	1.5	51489

表 5.4: 測定誤差（フレームレート 120fps）

平均測定誤差 [km/h]	平均処理時間 [ms]	最大誤差 [km/h]	最長処理時間 [ms]
1.56	3802	3.94	4924

すでに終わっていたため、測定実験をおこなうことができたのは 100km/h 前後のボールにとどまった。150km/h までのボールの測定には、3 月に開幕するプロ野球の試合等で確認が必要である。

第6章　まとめ

最新の携帯端末 iPhone6 上で動作するスポーツ用スピードガンアプリケーション iPhoneSG を開発した。

iPhonSG はフレームレート 240fps で撮影できる iPhone6 の性能をフルに利用したアプリケーションである。iPhone6 によりピッチャーの投球を横方面から撮影した映像を iPhone6 内で画像処理し、その球速をプラスマイナス 4km/h 未満の精度で推定できる。画像処理に要する時間は 30 秒程度である。

従来のスピードガン等による投球速度の測定がピッチャーとキャッチャーを結ぶ線のほぼ延長線上でスピードガンを構えなければならなっかったのに対し、iPhoneSG はベンチやコーチスボックスなどのグラウンド上、あるいは、観客席などからも測定が可能であり、スピードガンとしての利用可能域を大いに拡大する。

iPhopneSG による球速測定時にユーザーがおこなうべきことは、

1. ピッチャーとキャッチャーが同時に画面内に映る画角の調整、
2. 投球に合わせてボタンを押す

のふたつのみである。ボールの検出、追跡、球速算出等の処理はすべて iPhoneSG が自動でおこなう。

iPhoneSG により、野球ファンやアマチュア選手にとってコストや精度の面でハードルの高かった球速の測定をより身近で手軽なものにすることができる。

iPhoneSG は主に野球場面での利用を考えているが、他のスポーツ種目でも利用可能である。

iPhoneSG は近日中に Apple Store からの配布を計画中である。

参考文献

[1] Apple. Apple. https://developer.apple.com/xcode/.

[2] Smart Tools co. スピードガン:speedgun. https://play.google.com/store/apps/details?id=kr.sira.speed&hl=ja.

[3] Orta Therox Eloy Durn, Fabio Pelosin and many others. The CocoaPods Dev Team with contributions from many. cocoapods. http://cocoapods.org.

[4] GSMARENA. Gsmarena.com - gsm phone reviews, news, opinions, votes, manuals and more... http://www.gsmarena.com.

[5] Intel and Willow Garage. Opencv. http://opencv.org.

[6] KEINS. びゅん. https://itunes.apple.com/jp/app/id639458550.

[7] opencv dev team. Opencv 2.4.9.0 documentation. http://docs.opencv.org/genindex.html.

[8] The MacPorts Project. macports. http://www.macports.org.

[9] shuichi0526@gmail.com. Slow motion video recorder for ios. https://github.com/shu223/SlowMotionVideoRecorder.

[10] Christian Theobalt, Irene Albrecht, Jrg Haber, Marcus Magnor, and Hans-Peter Seidel. Pitching a baseball - tracking high-speed motion with multi-exposure images. *ACM Transactions on Graphics*, pp. 540–547, 2004.

[11] uenoma. Easy speedgun. https://itunes.apple.com/jp/app/easy-speedgun/id451039156.

[12] 一般財団法人全日本野球協会アマチュア野球規則委員会. 野球規則（抜粋）. http://asaka-aba.net/kisoku.html.

[13] ミズノ. Hp-2 取扱説明書. http://www.mizuno.co.jp/customer/pdf/2zm-1050.pdf.

[14] 蔭山雅洋, 和田智仁, 前田明. 野球投手における投球数の増加が投球速度と投球動作に及ほす影響. 日本体育学会大会予稿集 第 62 回大会, p. 149, 2011.

[15] 奥村浩正. 野球における投球動作の分析. Studies in health and sports science 創刊号, pp. 10–24, 1999.

[16] 蔭山雅洋, 岩本峰明, 杉山敬, 水谷未来, 金久博昭, 前田明. 大学野球投手における体幹の伸張-短縮サイクル運動および動作が投球速度に与える影響. 体育学研究 Vol. 59, pp. 189–201, 2014.

[17] 株式会社ウェブレッジ. スマートフォン・シェアランキング (top10)—株式会社ウェブレッジ. http://webrage.jp/mobile/data/sp_share.html.

[18] 株式会社マイナビ. Opencv2 プログラミングブック サポートサイト. http://book.mynavi.jp/support/pc/opencv2/.

[19] 斎藤健治, 仰木裕嗣, 井上伸一, 市川浩, 山岸正克, 宮地力, 高井省三. 手首で計測した加速度による投球スピードの推定. 体育学研究 第 47 巻 第 1 号, pp. 41–51, 2002.

[20] 勝亦陽一, 長谷川伸, 川上泰雄, 福永哲夫. 投球速度と筋力および筋量の関係. スポーツ科学研究 3, pp. 1–7, 2006.

[21] 島田一志, 川村卓. キューバ、アメリカおよび日本チーム投手のピッチング動作のキネマティクス的特徴一 2010 年世界大学野球選手権大会を対象として一. 日本体育学会大会予稿集 第 62 回大会, p. 149, 2011.

付録A　Appendix

A.1　AppDelegate

```
1  //
2  //  AppDelegate.h
3  //  SlowMotionVideoRecorder
4  //
5  //  Created by shuichi on 12/17/13.
6  //  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
7  //
8  //  MITライセンスの下、山口陽介が改変した。
9
10 #import <UIKit/UIKit.h>
11
12 @interface AppDelegate : UIResponder <UIApplicationDelegate>
13
14 @property (strong, nonatomic) UIWindow *window;
15
16 @end
```

```
1  //
2  //  AppDelegate.m
3  //  SlowMotionVideoRecorder
4  //
5  //  Created by shuichi on 12/17/13.
6  //  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
7  //
8  //  MITライセンスの下、山口陽介が改変した。
9
10 #import "AppDelegate.h"
11
12 @implementation AppDelegate
```

```
13
14 - (BOOL)application:(UIApplication *)application
       didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
15 {
16     // Override point for customization after application launch.
17     [UIApplication sharedApplication].statusBarOrientation =
           UIInterfaceOrientationLandscapeRight; //←ステータスバーも横向きに
           してやらないといけない
18     return YES;
19 }
20
21 - (void)applicationWillResignActive:(UIApplication *)application
22 {
23     // Sent when the application is about to move from active to
           inactive state. This can occur for certain types of temporary
            interruptions (such as an incoming phone call or SMS message
           ) or when the user quits the application and it begins the
           transition to the background state.
24     // Use this method to pause ongoing tasks, disable timers, and
           throttle down OpenGL ES frame rates. Games should use this
           method to pause the game.
25 }
26
27 - (void)applicationDidEnterBackground:(UIApplication *)application
28 {
29     // Use this method to release shared resources, save user data,
           invalidate timers, and store enough application state
           information to restore your application to its current state
           in case it is terminated later.
30     // If your application supports background execution, this
           method is called instead of applicationWillTerminate: when
           the user quits.
31 }
32
33 - (void)applicationWillEnterForeground:(UIApplication *)application
34 {
35     // Called as part of the transition from the background to the
           inactive state; here you can undo many of the changes made on
```

```
        entering the background.
36  }
37
38  - (void)applicationDidBecomeActive:(UIApplication *)application
39  {
40      // Restart any tasks that were paused (or not yet started) while
            the application was inactive. If the application was
            previously in the background, optionally refresh the user
            interface.
41  }
42
43  - (void)applicationWillTerminate:(UIApplication *)application
44  {
45      // Called when the application is about to terminate. Save data
            if appropriate. See also applicationDidEnterBackground:.
46  }
47
48  @end
```

A.2 ViewController

```
1   //
2   //  ViewController.h
3   //  SlowMotionVideoRecorder
4   //
5   //  Created by shuichi on 12/17/13.
6   //  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
7   //
8   //  MITライセンスの下、山口陽介が改変した。
9
10
11  #import <UIKit/UIKit.h>
12  #import <AVFoundation/AVFoundation.h>
13
14  @interface ViewController : UIViewController <
        AVCaptureVideoDataOutputSampleBufferDelegate>
15  @property (strong, nonatomic) AVCaptureDeviceInput *videoInput;
16  @property (strong, nonatomic) AVCaptureVideoDataOutput *
```

```
    videoDataOutput;
@property (strong, nonatomic) AVCaptureSession *session;
- (IBAction)setZoom:(id)sender;

@end
```

```
//
//  ViewController.mm
//  SlowMotionVideoRecorder
//
//  Created by shuichi on 12/17/13.
//  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
//
//  MITライセンスの下、山口陽介が改変した。

#import "ViewController.h"
#import "SVProgressHUD.h"
#import "AVCaptureManager.h"
#import <AssetsLibrary/AssetsLibrary.h>
#import <ios.h>
#import <opencv2/opencv.hpp>

@interface ViewController ()
<AVCaptureManagerDelegate>
{
    NSTimeInterval startTime;
    BOOL isNeededToSave;
}
@property (nonatomic, strong) AVCaptureManager *captureManager;
@property (nonatomic, assign) NSTimer *timer;
@property (nonatomic, strong) UIImage *recStartImage;
@property (nonatomic, strong) UIImage *recStopImage;
@property (nonatomic, strong) UIImage *outerImage1;
@property (nonatomic, strong) UIImage *outerImage2;

@property (nonatomic, weak) IBOutlet UILabel *statusLabel;
@property (nonatomic, weak) IBOutlet UISegmentedControl *fpsControl;
@property (nonatomic, weak) IBOutlet UIButton *recBtn;
@property (nonatomic, weak) IBOutlet UIImageView *outerImageView;
```

```
34 @property(nonatomic) BOOL enablesVideoStabilizationWhenAvailable;
35
36 @end
37
38
39 @implementation ViewController
40 CGFloat fps = 60.0;
41 CGFloat speed2 = 0.0;
42 CGFloat prev_x = 0.0;
43 CGFloat prev_y = 0.0;
44 CGFloat dif_x = 0.0;
45 CGFloat dif_y = 0.0;
46 CGFloat kakattajikan = 0.0;
47 int lost_ball = 0;
48 CGFloat min_dif_x = 10.0;
49
50 BOOL OSX = true;
51
52 typedef struct{
53     float speed;
54     float spenttime;
55 }Result;
56
57
58
59 - (void)showAlert:(NSString *)msg
60 {
61     [[[UIAlertView alloc] initWithTitle:msg
62                                 message:nil
63                                delegate:nil
64                       cancelButtonTitle:@"OK"
65                       otherButtonTitles:nil] show];
66 }
67
68 - (void)viewDidLoad
69 {
70     [super viewDidLoad];
71
```

```
72      self.captureManager = [[AVCaptureManager alloc]
            initWithPreviewView:self.view];
73      self.captureManager.delegate = self;
74
75
76      UIImage *image;
77      image = [UIImage imageNamed:@"ShutterButtonStart"];
78      self.recStartImage = [image imageWithRenderingMode:
            UIImageRenderingModeAlwaysTemplate];
79      [self.recBtn setImage:self.recStartImage
80                   forState:UIControlStateNormal];
81
82      image = [UIImage imageNamed:@"ShutterButtonStop"];
83      self.recStopImage = [image imageWithRenderingMode:
            UIImageRenderingModeAlwaysTemplate];
84
85      [self.recBtn setTintColor:[UIColor colorWithRed:245./255.
86                                                green:51./255.
87                                                 blue:51./255.
88                                                alpha:1.0]];
89      self.outerImage1 = [UIImage imageNamed:@"outer1"];
90      self.outerImage2 = [UIImage imageNamed:@"outer2"];
91      self.outerImageView.image = self.outerImage1;
92      [self.captureManager switchFormatWithDesiredFPS:60.0]; //初期
            FPSを設定
93  }
94
95
96  - (void)didReceiveMemoryWarning
97  {
98      [self showAlert:@"didReceiveMemoryWarning:memory␣warn"];
99      [super didReceiveMemoryWarning];
100 }
101
102
103 //
    ============================================================================
```

```
104 #pragma mark - Gesture Handler
105
106 - (void)handleDoubleTap:(UITapGestureRecognizer *)sender {
107
108     [self.captureManager toggleContentsGravity];
109 }
110
111
112 //
    ================================================================================

113 #pragma mark - Private
114
115
116 - (void)saveRecordedFile:(NSURL *)recordedFile {
117
118     [SVProgressHUD showWithStatus:@"saveRecordedFile..."
119                           maskType:SVProgressHUDMaskTypeGradient];
120
121     dispatch_queue_t queue = dispatch_get_global_queue(
        DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
122     dispatch_async(queue, ^{
123         ALAssetsLibrary *assetLibrary = [[ALAssetsLibrary alloc]
            init];
124         [assetLibrary writeVideoAtPathToSavedPhotosAlbum:
            recordedFile
125                                          completionBlock:
126          ^(NSURL *assetURL, NSError *error) {
127              dispatch_async(dispatch_get_main_queue(), ^{
128
129                  [SVProgressHUD dismiss];
130              });
131          }];
132     });
133 }
134
135
136
```

```
137 //
    ================================================================================

138 #pragma mark - Timer Handler
139
140 - (void)timerHandler:(NSTimer *)timer {
141
142     NSTimeInterval current = [[NSDate date] timeIntervalSince1970];
143     NSTimeInterval recorded = current - startTime;
144     if (recorded >2.0) {
145         [SVProgressHUD showWithStatus:@"Detecting..."
146                                 maskType:SVProgressHUDMaskTypeGradient
                                    ];
147         isNeededToSave = YES;
148
149         [self.captureManager stopRecording];
150
151         [self.timer invalidate];
152         self.timer = nil;
153
154
155         // change UI
156         [self.recBtn setImage:self.recStartImage
157                      forState:UIControlStateNormal];
158         self.fpsControl.enabled = YES;
159     }
160     self.statusLabel.text = [NSString stringWithFormat:@"%.2f",
        recorded];
161 }
162
163 // written by uehara.
164 - (void)showResult:(Result)result{
165     NSString *title;
166     NSString *message;
167     title = [ NSString stringWithFormat : @"Speed is ...\n %3.0f [km
        /h], %3.0f [mph] /n spent_time : %5.0f [ms]",result.speed,
        result.speed*0.62137, result.spenttime];
168     message = nil;
```

```
169     [self showAlert:title];
170 }
171
172 //
    ================================================================================

173 #pragma mark - AVCaptureManagerDeleagte
174
175 - (void)didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL
    error:(NSError *)error {
176     if (error) {
177         [self showAlert:@"URL_error"];
178         return;
179     }
180
181     if (!isNeededToSave) {
182         return;
183     }
184
185
186     [self saveRecordedFile:outputFileURL];
187     // 合成処理実行。動画ファイルのURLを渡す。
188     NSString *urlString = [outputFileURL absoluteString];
189     NSString *movieURL = [urlString substringFromIndex:8];
190
191     [self showResult:[self BallDitectFromMovie:movieURL]];
192
193 }
194
195 //ball_ditect
196 - (Result)BallDitectFromMovie:(NSString *)movieURL{
197     Result res;
198
199     double f = 1000.0/cv::getTickFrequency();
200     int64 time = cv::getTickCount();
201     int ball_cnt =0;
202     int cnt =0;
203     int ball_error=0;
```

```
204     float interval_pitch = 18.44;
205     float interval = interval_pitch * 830 / 1280;
206     [SVProgressHUD showWithStatus:@"BallDitectFromMovie..."
207                           maskType:SVProgressHUDMaskTypeGradient];
208
209     cv::VideoCapture capture;
210     if(capture.open(std::string([movieURL UTF8String]))){
211         cv::Mat frame[3];
212         cv::Mat dst_img;
213         for(int i=0; i<50;i++) capture >>dst_img;
214         while(1)
215         {
216             cv::Mat showImage;
217
218             if(cnt == 0){
219                 capture>>frame[0];
220
221                 if(frame[0].empty())
222                 {
223                     std::cout<<"Frame1Message->End of sequence 0"<<
                            std::endl;
224                     break;
225                 }
226                 capture>>frame[1];
227                 if(frame[1].empty())
228                 {
229                     std::cout<<"Frame1Message->End of sequence 1"<<
                            std::endl;
230                     break;
231                 }
232                 frame[2]=frame[0].clone(); //フレーム間差分
233                 frame[2]=frame[2]-frame[1];
234                 showImage = frame[2];
235                 cnt = 2;
236             }else if(cnt%2==0){
237                 capture>>frame[0];
238                 if(frame[0].empty())
239                 {
```

```
240                     std::cout<<"Frame1Message->End of sequence 1"<<
                            std::endl;
241                     break;
242                 }
243                 frame[2]=frame[1].clone();
244                 frame[2]=frame[2]-frame[0];
245                 showImage = frame[1];
246                 cnt++;
247             }else{
248                 capture>>frame[1];
249                 if(frame[1].empty())
250                 {
251                     std::cout<<"Frame2Message->End of sequence 2"<<
                            std::endl;
252                     break;
253                 }
254                 frame[2]=frame[0].clone();
255                 frame[2]=frame[2]-frame[1];
256                 showImage = frame[0];
257                 cnt++;
258             }
259
260             float roi_x = 0.0;
261             float roi_y = 0.0;
262             float roi_w = 0.0;
263             float roi_h = 0.0;
264
265             if (ball_cnt < 1) {
266                 roi_x = 1000.0;
267                 roi_y = 140.0;
268                 roi_w = 100.0;
269                 roi_h = 300.0;
270             } else {
271                 roi_x = prev_x - dif_x - 50.0;
272                 roi_y = prev_y - 50.0;
273                 roi_w = 100.0;
274                 roi_h = 100.0;
275             }
```

```
276             if (ball_cnt > 10 && prev_x < 170) {
277                 res.speed = interval*fps*3.6/ball_cnt;
278                 res.spenttime = (cv::getTickCount()-time)*f;
279                 return res;
280             }
281             if (!(0 <= roi_x && 0 <= roi_w && roi_x + roi_w <=
                    showImage.cols && 0 <= roi_y && 0 <= roi_h && roi_y +
                     roi_h <= showImage.rows)){
282                 [self showAlert:@"roi␣Error!"];
283                 res.spenttime = (cv::getTickCount()-time)*f;
284                 res.speed =0.0;
285                 return res;
286             }else{
287                 cv::Mat dst_img = frame[2].clone();
288                 cv::Mat roi_img(dst_img, cv::Rect(roi_x, roi_y,
                        roi_w, roi_h)); //x,y,w,h
289
290
291                 cv::cvtColor(roi_img, roi_img, CV_BGR2GRAY); //グレース
                        ケール変換
292                 cv::threshold(roi_img, roi_img, 10, 255, cv::
                        THRESH_BINARY); //二値
                        化
293
294                 cv::erode(roi_img, roi_img, cv::Mat(), cv::Point
                        (-1,-1), 2); //収
                        縮
295                 cv::dilate(roi_img, roi_img, cv::Mat(), cv::Point
                        (-1,-1), 2); //膨
                        張
296                 cv::erode(roi_img, roi_img, cv::Mat(), cv::Point
                        (-1,-1), 2);
297                 cv::dilate(roi_img, roi_img, cv::Mat(), cv::Point
                        (-1,-1), 3);
298
299                 // 輪郭を抽出
300                 // CV_RETR_EXTERNAL：最も外側の輪郭のみ抽出
301                 // CV_CHAIN_APPROX_NONE：輪郭の全部の点を出力
```

```
302                 cv::vector< cv::vector<cv::Point> > contours;
303                 cv::Mat contourImage(roi_img.size(), CV_8U, cv::
                        Scalar(255));
304                 cv::findContours(roi_img, contours, CV_RETR_EXTERNAL
                        , CV_CHAIN_APPROX_NONE);
305                 std::cout << "coutours_size"<<contours.size()<<std::
                        endl;
306
307                 if(contours.size() == 0 && ball_cnt > 2){
308                     prev_x = prev_x - min_dif_x;
309                     dif_x = min_dif_x;
310                     ball_cnt++;
311                     if (ball_error < 100) {
312                         ball_error++;
313                     } else {
314                         res.spenttime = (cv::getTickCount()-time)*f;
315                         res.speed =0.0;
316                         return res;
317                         [self showAlert:@"too many ball_error!"];
318                         res.spenttime = (cv::getTickCount()-time)*f;
319                         res.speed =0.0;
320                         return res;
321                     }
322                 }
323
324                 for(int i = 0; i < contours.size(); ++i) {
325
326                     size_t count = contours[i].size();
327
328                     if(count < 10 || count > 70) continue; // (小さ
                            すぎる|大きすぎる) 輪郭を除外
329
330                     cv::Point2d point;
331                     cv::Moments mom = cv::moments(contours[i]);
332                     point.x = mom.m10/mom.m00 + roi_x;
333                     point.y = mom.m01/mom.m00 + roi_y;
334
335                     if (ball_cnt < 2) {
```

```
336                         dif_x = 0.0;
337                     }else{
338                         dif_x = prev_x - point.x;
339                     }
340                     prev_x = point.x;
341                     prev_y = point.y;
342 
343                     ball_cnt++;
344                     NSLog(@"ball_cnt: %d",ball_cnt);
345                 }
346             }
347 
348             lost_ball = ball_error;
349         }
350         [self showAlert:@"Finish the ditection!"];
351         res.spenttime = (cv::getTickCount()-time)*f;
352         res.speed = interval*fps*3.6/ball_cnt;
353         return res;
354     }else{
355         [self showAlert:@"Failed to open"];
356     }
357     [self showAlert:@"finish the loop"];
358     res.spenttime = (cv::getTickCount()-time)*f;
359     res.speed =0.0;
360     return res;
361 
362 }
363 
364 
365 //
    ==============================================================================

366 #pragma mark - IBAction
367 
368 - (IBAction)recButtonTapped:(id)sender {
369 
370     // REC START
371     if (!self.captureManager.isRecording) {
```

```
372
373             // change UI
374             [self.recBtn setImage:self.recStopImage
375                          forState:UIControlStateNormal];
376             self.fpsControl.enabled = NO;
377
378             // timer start
379             startTime = [[NSDate date] timeIntervalSince1970];
380             self.timer = [NSTimer scheduledTimerWithTimeInterval:0.01
381                                                           target:self
382                                                         selector:
                                                             @selector(
                                                             timerHandler
                                                             :)
383                                                         userInfo:nil
384                                                          repeats:YES];
385             [self.captureManager startRecording];
386         }
387         // REC STOP
388         else {
389         }
390     }
391
392
393     - (IBAction)fpsChanged:(UISegmentedControl *)sender {
394
395         // Switch FPS
396         CGFloat desiredFps = 60.0;;
397         switch (self.fpsControl.selectedSegmentIndex) {
398             case 0:
399             default:
400             {
401                 desiredFps = 60.0;
402                 break;
403             }
404             case 1:
405                 desiredFps = 120.0;
406                 break;
```

```
407             case 2:
408                 desiredFps = 240.0;
409
410                 break;
411         }
412
413         fps = desiredFps;
414         [SVProgressHUD showWithStatus:@"Switching..."
415                               maskType:SVProgressHUDMaskTypeGradient];
416
417         dispatch_queue_t queue = dispatch_get_global_queue(
                DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
418         dispatch_async(queue, ^{
419
420             if (desiredFps > 0.0) {
421                 [self.captureManager switchFormatWithDesiredFPS:
                        desiredFps];
422             }
423             else {
424                 [self.captureManager resetFormat];
425             }
426
427             dispatch_async(dispatch_get_main_queue(), ^{
428
429                 if (desiredFps > 60.0) {
430                     self.outerImageView.image = self.outerImage2;
431                 }
432                 else {
433                     self.outerImageView.image = self.outerImage1;
434
435                 }
436                 [SVProgressHUD dismiss];
437             });
438         });
439 }
440
441
442 - (IBAction)setZoom:(id)sender {
```

```
443     UISlider *_slider = sender;
444     AVCaptureDevice* videoCaptureDevice = [AVCaptureDevice
            defaultDeviceWithMediaType:AVMediaTypeVideo];
445     NSError* error = nil;
446     if ([videoCaptureDevice lockForConfiguration:&error]) {
447
448         [videoCaptureDevice rampToVideoZoomFactor:_slider.value
                //UISlider
449
450                                                 withRate:5.0];
                                                                //ズーム速
                                                    度
451
452         [videoCaptureDevice unlockForConfiguration];
453     } else {
454         NSlog(@"%s [ERROR] %@", __PRETTY_FUNCTION__, error);
455     }
456
457 }
458
459
460
461 @end
```

A.3 AVCaptureManager

```
1  //
2  //  AVCaptureManager.h
3  //  SlowMotionVideoRecorder
4  //  https://github.com/shu223/SlowMotionVideoRecorder
5  //
6  //  Created by shuichi on 12/17/13.
7  //  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
8  //
9  //  MITライセンスの下、山口陽介が改変した。
10
11 #import <Foundation/Foundation.h>
12 #import <AVFoundation/AVFoundation.h>
```

```
13
14 @protocol AVCaptureManagerDelegate <NSObject>
15 - (void)didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL
16                                        error:(NSError *)error;
17 @end
18
19
20 @interface AVCaptureManager : NSObject
21
22 @property (nonatomic, assign) id<AVCaptureManagerDelegate> delegate;
23 @property (strong, nonatomic) AVCaptureVideoDataOutput *
       videoDataOutput;
24 @property (nonatomic, readonly) BOOL isRecording;
25
26
27 - (id)initWithPreviewView:(UIView *)previewView;
28 - (void)toggleContentsGravity;
29 - (void)resetFormat;
30 - (void)switchFormatWithDesiredFPS:(CGFloat)desiredFPS;
31 - (void)startRecording;
32 - (void)stopRecording;
33 @end
```

```
1  //
2  //  AVCaptureManager.m
3  //  SlowMotionVideoRecorder
4  //  https://github.com/shu223/SlowMotionVideoRecorder
5  //
6  //  Created by shuichi on 12/17/13.
7  //  Copyright (c) 2013 Shuichi Tsutsumi. All rights reserved.
8  //
9  //  MITライセンスの下、山口陽介が改変した。
10 #import "AVCaptureManager.h"
11 #import <AVFoundation/AVFoundation.h>
12 #import <ios.h>
13 #import <opencv2/opencv.hpp>
14
15 @interface AVCaptureManager ()
16 <AVCaptureFileOutputRecordingDelegate>
```

```
17 {
18     CMTime defaultVideoMaxFrameDuration;
19 }
20 @property (nonatomic, strong) AVCaptureSession *captureSession;
21 @property (nonatomic, strong) AVCaptureMovieFileOutput *fileOutput;
22 @property (nonatomic, strong) AVCaptureDeviceFormat *defaultFormat;
23 @property (nonatomic, strong) AVCaptureVideoPreviewLayer *
       previewLayer;
24
25 @end
26
27
28 @implementation AVCaptureManager
29
30 UIImage *image = nil;
31 UIImage *image2 = nil;
32 cv::Mat src;
33 cv::Mat src2;
34 int cnt = 0;
35 int ball_cnt = 0;
36 int unditect_cnt = 0;
37 int ditect_flag = 0;
38 int direction_x = 0;
39 int direction_y = 0;
40 int direction_x2 = 0;
41 int direction_y2 = 0;
42 int a = 0;
43 int b = 0;
44 int c = 480;
45 int d = 360;
46
47 - (id)initWithPreviewView:(UIView *)previewView {
48
49     self = [super init];
50
51     if (self) {
52
53         NSError *error;
```

```
54
55          self.captureSession = [[AVCaptureSession alloc] init];
56          self.captureSession.sessionPreset =
                AVCaptureSessionPresetInputPriority;
57
58          AVCaptureDevice *videoDevice = [AVCaptureDevice
                defaultDeviceWithMediaType:AVMediaTypeVideo];
59          AVCaptureDeviceInput *videoIn = [AVCaptureDeviceInput
                deviceInputWithDevice:videoDevice error:&error];
60
61          if (error) {
62              NSLog(@"Video␣input␣creation␣failed");
63              return nil;
64          }
65
66          if (![self.captureSession canAddInput:videoIn]) {
67              NSLog(@"Video␣input␣add-to-session␣failed");
68              return nil;
69          }
70          [self.captureSession addInput:videoIn];
71
72
73          // save the default format
74          self.defaultFormat = videoDevice.activeFormat;
75          defaultVideoMaxFrameDuration = videoDevice.
                activeVideoMaxFrameDuration;
76
77
78          AVCaptureDevice *audioDevice= [AVCaptureDevice
                defaultDeviceWithMediaType:AVMediaTypeAudio];
79          AVCaptureDeviceInput *audioIn = [AVCaptureDeviceInput
                deviceInputWithDevice:audioDevice error:&error];
80          [self.captureSession addInput:audioIn];
81          // 入力と出力からキャプチャーセッションを作成
82
83          self.fileOutput = [[AVCaptureMovieFileOutput alloc] init];
84          [self.captureSession addOutput:self.fileOutput];
85
```

```
86          self.previewLayer = [[AVCaptureVideoPreviewLayer alloc]
                initWithSession:self.captureSession];
87          self.previewLayer.frame = previewView.bounds;
88          self.previewLayer.contentsGravity =
                kCAGravityResizeAspectFill;
89          self.previewLayer.videoGravity =
                AVLayerVideoGravityResizeAspectFill;
90          [previewView.layer insertSublayer:self.previewLayer atIndex
                :0];
91          [[self.previewLayer connection]setVideoOrientation:
                AVCaptureVideoOrientationLandscapeRight]; //横向き
                に
92          [self.captureSession startRunning];
93      }
94      return self;
95  }
96
97
98
99  //
        ============================================================================

100 #pragma mark - Public
101
102 - (void)toggleContentsGravity {
103
104     if ([self.previewLayer.videoGravity isEqualToString:
            AVLayerVideoGravityResizeAspectFill]) {
105
106         self.previewLayer.videoGravity =
                AVLayerVideoGravityResizeAspect;
107     }
108     else {
109         self.previewLayer.videoGravity =
                AVLayerVideoGravityResizeAspectFill;
110     }
111 }
112
```

```
113 - (void)resetFormat {
114
115     BOOL isRunning = self.captureSession.isRunning;
116
117     if (isRunning) {
118         [self.captureSession stopRunning];
119     }
120
121     AVCaptureDevice *videoDevice = [AVCaptureDevice
            defaultDeviceWithMediaType:AVMediaTypeVideo];
122     [videoDevice lockForConfiguration:nil];
123     videoDevice.activeFormat = self.defaultFormat;
124     videoDevice.activeVideoMaxFrameDuration =
            defaultVideoMaxFrameDuration;
125     [videoDevice unlockForConfiguration];
126
127     if (isRunning) {
128         [self.captureSession startRunning];
129     }
130 }
131
132 - (void)switchFormatWithDesiredFPS:(CGFloat)desiredFPS
133 {
134     BOOL isRunning = self.captureSession.isRunning;
135
136     if (isRunning)  [self.captureSession stopRunning];
137
138     AVCaptureDevice *videoDevice = [AVCaptureDevice
            defaultDeviceWithMediaType:AVMediaTypeVideo];
139     AVCaptureDeviceFormat *selectedFormat = nil;
140     int32_t maxWidth = 0;
141     AVFrameRateRange *frameRateRange = nil;
142
143     for (AVCaptureDeviceFormat *format in [videoDevice formats]) {
144
145         for (AVFrameRateRange *range in format.
                videoSupportedFrameRateRanges) {
146
```

```
147             CMFormatDescriptionRef desc = format.formatDescription;
148             CMVideoDimensions dimensions =
                    CMVideoFormatDescriptionGetDimensions(desc);
149             int32_t width = dimensions.width;
150
151             if (range.minFrameRate <= desiredFPS && desiredFPS <=
                    range.maxFrameRate && width >= maxWidth) {
152
153                 selectedFormat = format;
154                 frameRateRange = range;
155                 maxWidth = width;
156             }
157         }
158     }
159
160     if (selectedFormat) {
161
162         if ([videoDevice lockForConfiguration:nil]) {
163
164             NSLog(@"selected format:%@", selectedFormat);
165             videoDevice.activeFormat = selectedFormat;
166             videoDevice.activeVideoMinFrameDuration = CMTimeMake(1,
                    (int32_t)desiredFPS);
167             videoDevice.activeVideoMaxFrameDuration = CMTimeMake(1,
                    (int32_t)desiredFPS);
168             [videoDevice unlockForConfiguration];
169         }
170     }
171
172     if (isRunning) [self.captureSession startRunning];
173 }
174
175 - (void)startRecording {
176
177     NSDateFormatter* formatter = [[NSDateFormatter alloc] init];
178     [formatter setDateFormat:@"yyyy-MM-dd-HH-mm-ss"];
179     NSString* dateTimePrefix = [formatter stringFromDate:[NSDate
            date]];
```

```
180
181     int fileNamePostfix = 0;
182     NSArray *paths = NSSearchPathForDirectoriesInDomains(
            NSDocumentDirectory, NSUserDomainMask, YES);
183     NSString *documentsDirectory = [paths objectAtIndex:0];
184     NSString *filePath = nil;
185     do
186         filePath =[NSString stringWithFormat:@"/%@/%@-%i.mp4",
                documentsDirectory, dateTimePrefix, fileNamePostfix++];
187     while ([[NSFileManager defaultManager] fileExistsAtPath:filePath
            ]);
188
189     NSURL *fileURL = [NSURL URLWithString:[@"file://"
            stringByAppendingString:filePath]];
190     [self.fileOutput startRecordingToOutputFileURL:fileURL
            recordingDelegate:self];
191 }
192
193 - (void)stopRecording {
194
195     [self.fileOutput stopRecording];
196 }
197
198
199
200 //
    ================================================================================

201 #pragma mark - AVCaptureFileOutputRecordingDelegate
202
203 - (void)                    captureOutput:(AVCaptureFileOutput *)
    captureOutput
204     didStartRecordingToOutputFileAtURL:(NSURL *)fileURL
205                         fromConnections:(NSArray *)connections
206 {
207     _isRecording = YES;
208     NSLog(@"started");
209 }
```

```
210
211 - (void)                    captureOutput:(AVCaptureFileOutput *)
        captureOutput
212     didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL
213                             fromConnections:(NSArray *)connections error
                                    :(NSError *)error
214 {
215     _isRecording = NO;
216     NSLog(@"finished");
217
218
219
220     NSString *urlString = [outputFileURL absoluteString];
221     NSString *movieURL = [urlString substringFromIndex:8];
222     NSLog(@"movieURL:%@", movieURL);
223
224     if ([self.delegate respondsToSelector:@selector(
            didFinishRecordingToOutputFileAtURL:error:)]) {
225         [self.delegate didFinishRecordingToOutputFileAtURL:
                outputFileURL error:error];
226     }
227 }
228
229
230
231 @end
```